ANA-TD-001

Spectrum Analyser Technical Design Document

A real-time audio spectrum analyser instrument delivered as a browser-based web application. This document records how it works, why it is built the way it is and the digital signal processing decisions behind the display.

Reference	ANA-TD-001
Title	Spectrum Analyser Technical Design
Version	1.0
Date	30 August 2026
Author	MP
System version described	v0.9.0
Status	Issued

Contents

1. Purpose and scope

The Spectrum Analyser is a full-viewport, real-time audio analysis instrument delivered as a private browser-based application. It renders the frequency content of a live audio signal as an animated display in the browser, styled as a piece of physical audio hardware rather than a web page.

This document explains the system at v0.9.0: the architecture, the Web Audio processing pipeline, the mathematics of mapping FFT output onto a musically useful display, the render engines and the persistence model. It is the third document in the technical series alongside PEM-TD-001 and the GPU Maximiser technical document, and it differs from both in one structural way: the analyser has no server-side application state. Audio analysis and display processing are performed in the browser, and no analysis data is retained or transmitted by the application. The document is therefore primarily an engineering note on real-time DSP in a browser rather than an operations record.

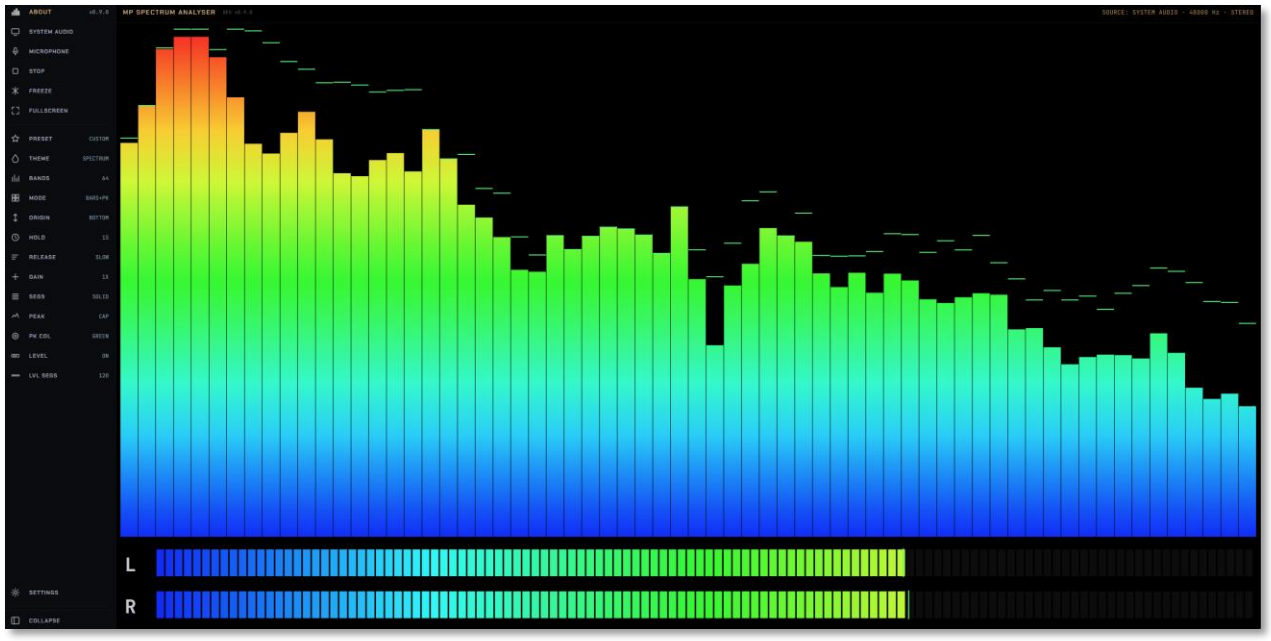
The instrument was built rather than bought. Hardware spectrum analysers of the sort this emulates are either vintage units at collector prices or software plug-ins locked inside a DAW. A browser build provides the same style of responsive display on ordinary web-capable machines without requiring a dedicated hardware unit.

2. System overview

The analyser presents as a single full-viewport instrument panel. The primary display is a bank of frequency bands animated at the display refresh rate, fed by genuine spectral data from the Web Audio API rather than a canned animation. Around the display sit the instrument controls:

- Ten colour themes covering the display and panel chrome.
- Six presets giving one-touch combinations of display settings.
- Three render styles changing how the band data is painted.
- A collapsible settings rail for live adjustment without leaving the display.
- A Settings modal including JSON backup and restore of the full configuration, so a tuned setup can be exported and restored to another browser.

All configuration is client side and persists locally. The delivery layer serves the application and enforces access; after initial delivery, signal processing and display activity are handled in the browser.



Screenshot of MP Spectrum Analyser

3. Architecture

The application is packaged as a compact browser-based web deployment. The presentation, styles and client-side scripts are delivered together, keeping the runtime simple and self-contained. Deployment-specific hosting, routing and access-control implementation are intentionally outside the scope of this document.

There is deliberately no application database or server-side persistence. The analyser processes a live signal and discards the analysis data as it draws it; persistence needs are limited to user preferences, which localStorage covers. This keeps the runtime lightweight and reduces the number of components involved.

The client is similarly self-contained. The page declares a dark colour scheme in the head so the first frame paints dark with no white flash on refresh, and the display is rendered to canvas rather than DOM elements, for reasons covered in section 8.

4. Access control

The analyser is a private tool. Access is restricted at the delivery layer and unauthorised requests are rejected before the instrument is served. The specific authentication, request-handling and monitoring controls are intentionally not documented here; they are maintained separately from this design document.

Additional pre-release access restrictions may be applied while the tool remains private. Any future public release would be preceded by a separate security and publication review.

5. The audio pipeline

The Web Audio API models audio as a directed graph of nodes. The analyser's graph is short and is created once, on the first user gesture, because browsers refuse to start an AudioContext without one. This is an autoplay-policy requirement, not a design choice, and the instrument is built around it: the display idles until the user acts, then the graph is constructed and the render loop starts.

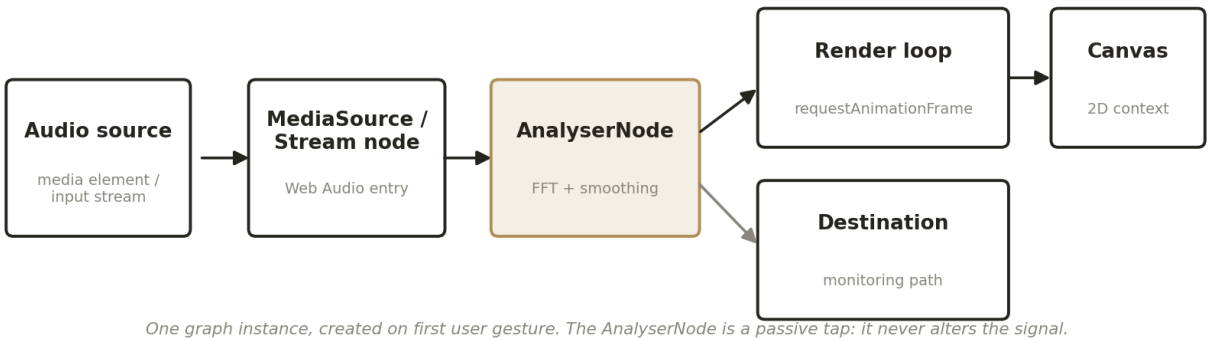


Figure 1. The Web Audio processing graph. The AnalyserNode taps the signal without modifying it; the render loop pulls spectral data from the node once per displayed frame.

The centre of the graph is the AnalyserNode, which performs a windowed fast Fourier transform over the most recent audio samples and exposes the result as an array of magnitude values, one per frequency bin. Two of its properties define the instrument's character and are worth understanding properly: fftSize, which sets the trade between frequency resolution and time responsiveness (section 6), and smoothingTimeConstant, which sets how the display moves (section 7).

A key architectural point: the render loop is pull-based, not push-based. Audio processing happens on the audio thread at the sample rate; the display runs on the main thread via requestAnimationFrame at the monitor's refresh rate. The AnalyserNode is the bridge between the two clocks. Each displayed frame simply asks the node for the current spectrum, so audio never waits for rendering and a dropped display frame costs nothing but smoothness.

6. Frequency analysis and band mapping

6.1 The resolution trade

An FFT of size N over audio sampled at rate fs produces N/2 usable bins, each covering fs/N hertz. Bigger transforms mean finer frequency detail but a longer analysis window, so the display responds more slowly to change. Figure 2 shows the relationship for the two sample rates a browser will realistically hand over.

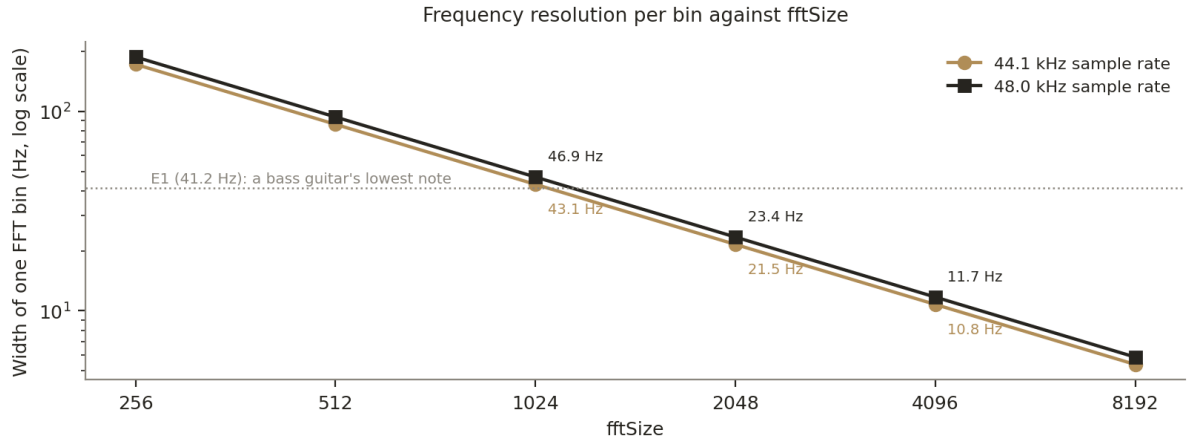


Figure 2. Bin width against fftSize. The dotted line marks the lowest note of a bass guitar: below an fftSize of roughly 2048 an entire bass line falls inside one or two bins.

6.2 Linear bins, logarithmic ears

FFT bins are spaced linearly in frequency, but human hearing and musical structure are logarithmic: every octave doubles the frequency. A display that simply drew the bins as bars would spend nearly all its width on the top two octaves and squeeze everything musical into the leftmost sliver. Every credible analyser therefore maps linear bins onto logarithmically spaced display bands, and this mapping is the real engineering in the instrument.

Band edges are placed geometrically between the display's lower and upper frequency limits, then each band claims the FFT bins whose centre frequencies fall inside it. The consequence is shown in figure 3: at the top of the range a single band aggregates over a hundred bins, while at the bottom several bands compete for one or two bins each.

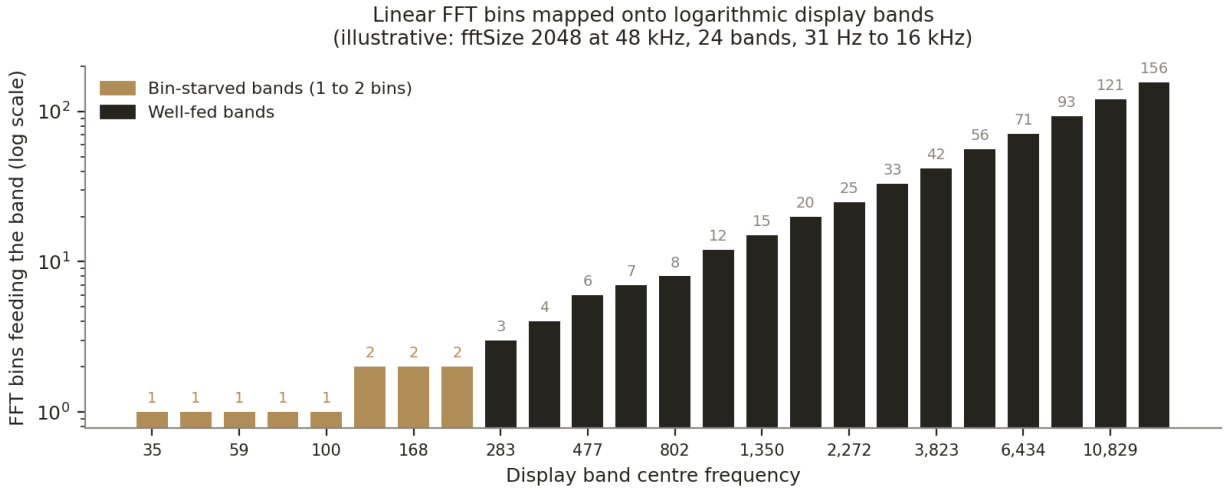


Figure 3. Bins available to each display band under an illustrative 24-band layout. The brass bars mark bin-starved bands at the bass end, where careful handling decides whether the display looks professional or broken.

Both ends need deliberate treatment. At the top, a band's value is reduced from its many contributing bins; taking the peak keeps transients visible where averaging would flatten them into mush. At the bottom, bands sharing a bin must interpolate between neighbouring bins rather than duplicate them, otherwise adjacent bass bands move in lockstep and the fraud is visible to anyone who has watched a real analyser. The bottom edge of the display range is

also chosen to sit where the FFT still has something honest to say, which is why the range does not extend to 20 Hz just because a marketing sheet would like it to.

7. Temporal smoothing

Raw FFT magnitudes flicker violently frame to frame. The AnalyserNode applies exponential smoothing between successive analysis frames, governed by smoothingTimeConstant: each new frame is blended with the previous smoothed frame, with the constant setting the blend. The behaviour is shown in figure 4 for a fixed input burst.

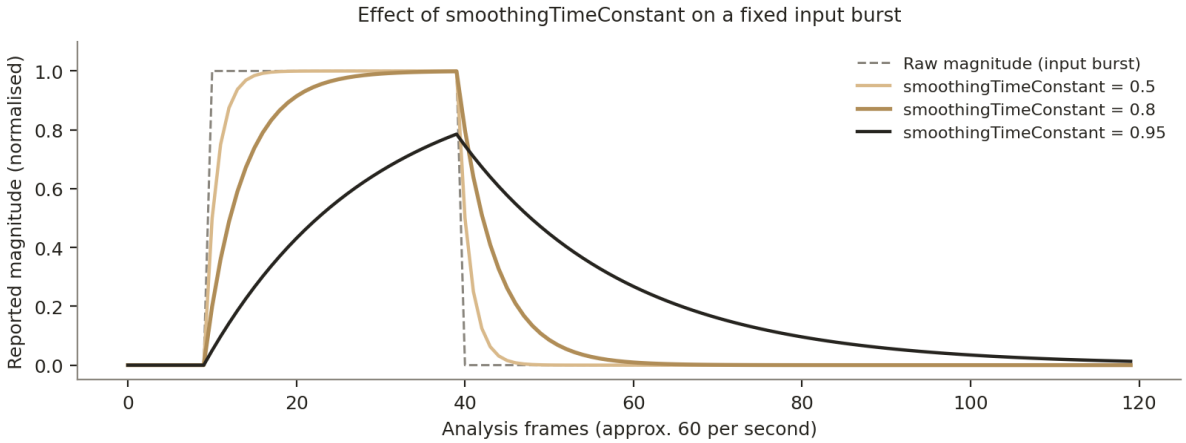


Figure 4. The same input burst reported at three smoothing settings. Low values track the signal but shimmer; high values glide but smear attacks and hold energy on screen after the music has moved on.

The choice is an aesthetic one disguised as a technical one. Values near 0.5 feel nervous and digital; values above 0.9 turn drum hits into slow waves. The analyser sits in the band around 0.8 where the display reads as an instrument with pleasing ballistics rather than either an oscilloscope or a lava lamp, and per-style adjustments on top of the node's own smoothing (for example peak-hold behaviour and decay rates) are applied in the render layer where they can differ between render styles without touching the audio graph.

8. Render styles and the frame budget

The display is painted to a canvas 2D context inside a requestAnimationFrame loop. Canvas was chosen over DOM animation deliberately: a band display implies dozens of elements updating sixty times a second, and doing that with styled DOM nodes forces layout and paint work the browser cannot batch efficiently. One canvas, cleared and redrawn each frame, keeps the whole display inside a single composited surface.

Three render styles ship in v0.9.0, selectable live from the settings rail. Each is a different painter over the same band data, which is the cheap and correct way to offer visual variety: the analysis pipeline is shared and only the final drawing differs. The frame budget at 60 Hz is 16.7 ms, and the analyser's frame cost is dominated by fill operations rather than the FFT read, so the styles are designed to keep per-frame fill area bounded. The instrument holds its refresh rate comfortably on representative desktop hardware, including integrated graphics.

Resize handling re-derives all display geometry from the viewport rather than storing pixel positions, so the instrument is correct at any window size, and the canvas backing store is scaled to devicePixelRatio so the display stays sharp on high-density screens instead of rendering soft and being stretched.

9. Theming, presets and persistence

Ten colour themes style the display and the surrounding panel. Themes are data, not code: each is a small set of colour values consumed by the painters and the CSS, so adding a theme is an editorial decision rather than an engineering one. Six presets sit above the themes and capture complete display configurations for one-touch recall.

All user state persists in localStorage under the instrument's own keys. The Settings modal can export the full configuration as a JSON document and restore it, which serves two purposes: moving a tuned setup between browsers and keeping an optional backup of user preferences. Restore validates the incoming document rather than trusting it, since a JSON file is user input like any other.

10. Runtime configuration reference

The table below records the analyser's principal runtime constants and where they live. Values marked for confirmation are to be read from the v0.9.0 implementation and recorded in the next issue of this document; they are deliberately not quoted from memory here, because a technical document that guesses is worse than one that says so.

Constant	Role	Value at v0.9.0
fftSize	FFT window size; sets bin resolution (figure 2)	Confirm against implementation
smoothingTimeConstant	AnalyserNode inter-frame smoothing (figure 4)	Confirm against implementation
Band count	Number of logarithmic display bands	Confirm against implementation
Display range	Lower and upper band edge frequencies	Confirm against implementation
minDecibels / maxDecibels	Magnitude range mapped to display height	Confirm against implementation
Themes	Colour themes available	10
Presets	One-touch display configurations	6
Render styles	Selectable display painters	3
Persistence	Client-side storage of user configuration	localStorage + JSON export/restore

Table 1. Runtime configuration reference. Figures in this document that depend on these constants (figures 2 and 3) are labelled as illustrative and show the mathematics at representative values.

11. Known limitations

- Sample rate is the browser's choice. The AudioContext reports 44.1 or 48 kHz depending on platform, and bin widths shift accordingly. The band mapping derives from the reported rate rather than assuming one, but the two platforms are not bit-identical.
- Bass resolution is bounded by physics. Below the bin-starved region shown in figure 3 the display is interpolating, honestly, but interpolating. A larger fftSize would sharpen the bass at the cost of display responsiveness, and v0.9.0 sits where the instrument feels best rather than where the numbers are largest.
- The AnalyserNode's FFT windowing and smoothing are implementation-defined in the small print. Behaviour is consistent within a browser but the specification leaves headroom, so cross-browser output can differ subtly at identical settings.
- The instrument is desktop-first. It functions on mobile but the instrument-panel layout is designed around a landscape viewport and a pointer.
- No audio is recorded, stored or transmitted anywhere by design, which also means there is no capture facility: the analyser shows the signal, it does not keep it.

12. Roadmap and parked work

Parked at v0.9.0, in no committed order:

- Public release preparation and checklist, including security, privacy and metadata review if the tool is opened up.
- Waterfall, oscilloscope and goniometer display engines, extending the instrument beyond band display into time-frequency, time-domain and stereo-field views.
- A custom colour picker alongside the ten built-in themes.
- Save-your-own presets alongside the six built-in ones.
- A follow-up issue of this document with the configuration constants of table 1 confirmed from source.

The waterfall, oscilloscope and goniometer engines are the substantial items. All three read from the same audio graph, so they are render-layer work: the waterfall needs a scrolling history buffer, the oscilloscope reads time-domain data the AnalyserNode already exposes, and the goniometer needs the stereo channels split ahead of analysis. None requires architectural change, which is the benefit of having kept the analysis pipeline and the painters separate from the start.

13. Document history

Version	Date	Author	Change
1.0	30 August 2026	MP	First issue, describing the analyser at v0.9.0. Configuration constants in table 1 marked for confirmation from source at next issue.